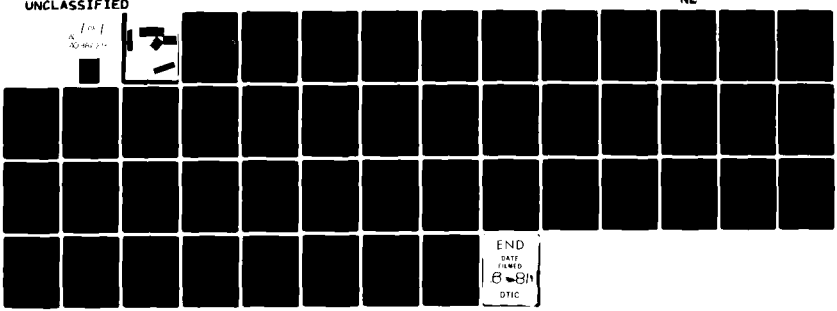
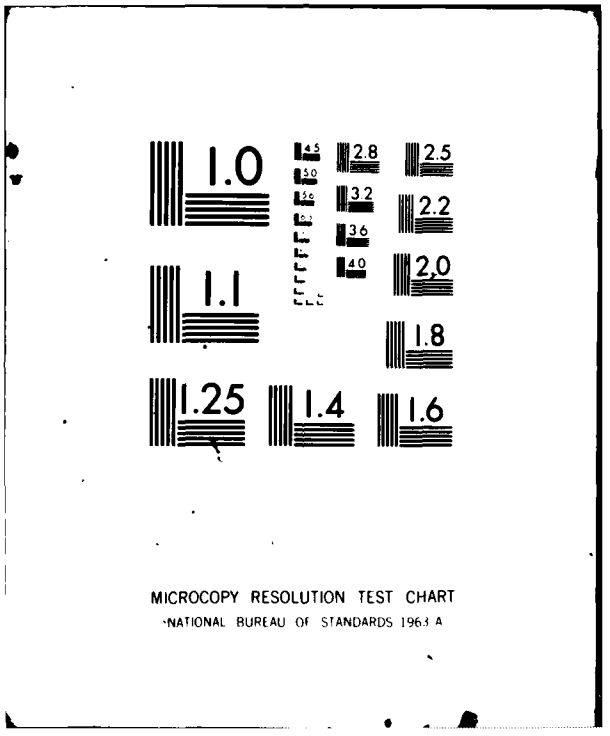
AD-A098 629 WISCONSIN UNIV-LA CROSSE RIVER STUDIES CENTER F/G 8/8
REGRESSION SIMULATION MODEL. APPENDIX X. USERS MANUAL.(U)
MAR 81 T O CLAFLIN, R G RADA DACW25-78-C-0047
UNCLASSIFIED NL



END
DATE FILMED
8-81
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

LEVEL II
C

APPENDIX X

(12) 48

(1)

(6) REGRESSION SIMULATION MODEL - Appendix X.

USERS MANUAL

Submitted to

The Great River Environmental Action Team

(Region II)

Contract No. DACW25-78-C-0047

(15)

by

(10) Thomas O. Claflin

Ronald G. Rada

The River Studies Center

University of Wisconsin-La Crosse

La Crosse, Wisconsin 54601

(11) March 1981

TABLE OF CONTENTS

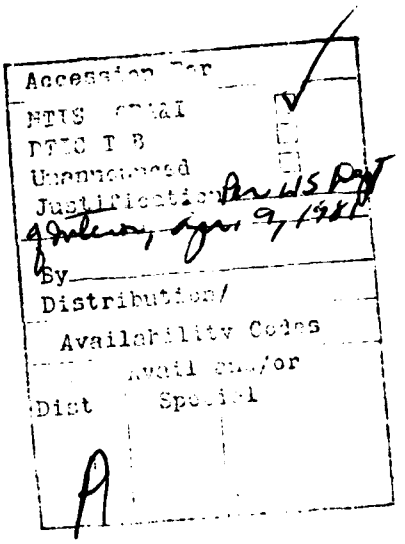
INTRODUCTION 1

APPENDIX I. Regression Model Trial Run 13

APPENDIX II. Field and Laboratory Methods 15

APPENDIX III. Understanding Basic Language and the Terminal . . . 18

APPENDIX IV. Pertinent Literature 42



INTRODUCTION

The application of mathematical models to ecological systems is difficult because of the complex interactions between the biological systems and the physical-chemical components of the ecosystems. It is appropriate to state that the biological regimes are strongly affected by their surroundings. It is also appropriate to assume that if given enough time, the biological systems will approach a state of equilibrium with their environment regarding energy flow, population dynamics and nutrient cycling.

When observing an ecosystem one can recognize several features which include:

(1) Energy traffic with a loss of energy at consecutively higher trophic levels;

(2) Interactions between and among populations; and

(3) Interactions between populations and their environment.

Finally it should be noted that most of the ecosystems that we observe are not in true equilibrium. The stability of ecosystems varies considerably, but even the most stable systems still change. Most of the aquatic systems that we observe are proceeding toward higher states of eutrophy. Because of these interactions and degrees of stability, it is virtually impossible to describe the systems with strict mathematical expressions, and descriptions of entire ecosystems are beyond the capabilities of even the most complex computers. Limited descriptions of ecosystems, however, can be made because many interactions are readily apparent to the observer.

Other relationships which are perhaps more subtle can be revealed through the use of statistical manipulations such as correlation and regression analyses. These and others have been applied to a large number of data points describing a navigation pool on the Upper Mississippi River, and have been employed to determine which physical and chemical parameters are important in controlling the densities and distributions of selected organisms in the river.

In developing this regression model system the investigators made several assumptions regarding predictability:

1. That the population of any given species is controlled by a set of environmental parameters.
2. That environment densities change with a change of the trophic status of the environment.
3. That the aquatic system is proceeding toward a higher trophic status.

The first two assumptions are no doubt valid. These relationships are embodied in the principles of general ecology. The third assumption is also probably valid since the entirety of the Upper Mississippi River is impounded into a series of shallow navigation pools which are governed by reservoir dynamics. Upon acceptance of these assumptions, it becomes conceivable that with careful selection, a continuum of specific habitat types can be established with the ecosystem. The difference in population densities within these habitat types would represent organismal responses to eutrophication (responses being changes in natality, mortality, and dispersion). It is also conceivable that once the relationships between organisms and their environment become established, one could apply the numerical relationships to other new ecological areas, provided the ranges of the new physical and chemical parameters are within those established in the

initial data base. The actual development of the model involves two basic steps. The first step is to establish the simple regression coefficients between all biological and physical/chemical parameters. The second step is to perform a multiple correlation and regression analysis and to establish a prediction equation (model). The prediction equation contains the partial regression coefficients (B-weights) which are the individual multiplicative factors used for prediction. The predicted value for any dependent variable therefore is the accumulative product of the specific B-weights and the values of the appropriate predictors (independent variables).

The regression model, then, is mathematically simple and is based on regression relationships determined using empirical data from samples. The following inherent limitations exist in this type of model, however:

1) Limitation in space; The model is likely to be limited to a defined reach of the Mississippi River as it is affected by latitude and man's activities.

2) Limitation in time; The model will only predict the final outcomes at a new presumed point of equilibrium. It has no capability to predict the rates of changes due to perturbations.

3) Limitation on ranges of predictor parameters (independent variables); The relationships determined between the predictors and the prediction are empirical. Thus, there is no basis for the assumption that the relationships can be extended beyond the ranges that were used in building the original data base, i.e. the regression equations may not be linear.

4) Inability to predict precise values because the model predicts the "most probable value."

TO WHAT ECOLOGICAL SITUATIONS CAN I APPLY THIS SYSTEM WITH SOME ASSURANCE OF SUCCESS?

The regression model can ostensibly be applied to several management situations that might involve changes in:

1. Current velocity.
2. Water depth.
3. Sediment particle size regime.
4. Organic content of the sediments and the attendant changes in sediment nitrogen and phosphorus.

These applications translated into man's activities on the Upper Mississippi River would include:

1. Channel dredging and associated spoiling in lateral areas.
2. Side-channel openings or culvert construction with associated changes in recipient areas of flow.
3. Occulusion of side channels which promote eutrophication in lateral water areas.

The changes associated with the above activities are probably embraced in most management activities on the Upper Mississippi River. Any other activities that would involve the biological changes could be included as a possible area of application for such a model.

For example, you are faced with the following problem: Within your management area is a shallow pond that has been completely isolated from the channel, and consequently receives no flow of water through it. A desire to divert a fresh flow of water through it has been expressed, and you wish to determine with the model what the biological outcome will be. The model could ostensibly predict those outcomes provided that you supply the computer with enough information concerning estimates of the projected physical/chemical change.

Here is the table that is provided when the operator asks for the parameter to be predicted:

Table 1. Code numbers for each dependent variable that can be predicted by the model.

NUMBER CODE	PARAMETER
BENTHIC INVERTEBRATES	
ORGANISMS (NUMBER PER SQUARE METER)	
1	NEMATODA
2	OLIGOCHAETA
3	HIRUDINEA
4	Hyallela azteca
5	Cheumatopsyche
6	Hexagenia
7	Chironomus
8	Coelotanypus
9	Polypedilum
10	Pentaneura
11	Palpomyia
12	Musculium
13	Sphaerium
14	Oecetis
15	Total Benthic Biomass (GM/SQ Meter)
MACROPHYTES	
16	Ceratophyllum demersum
17	Sagittaria latifolia
18	Nuphar variegatum
19	Potamogeton nodosus
20	Sagittaria rigida
21	Potamogeton pectinatus
22	Potamogeton foliosus
23	Heterantheria dubia
24	Mean Total Biomass (GM/SQ Meter, Dry Wt.)
FISH (% NUMBER)	
25	Cyprinus carpio
26	Ictalurus punctatus
27	Aplodinotus grunniens
28	Dorosoma cepedianum
29	Micropterus salmoides
30	Esox lucius
31	Ambloplites rupestris
32	Maxostoma macrolepidatum
33	Micropterus dolomieui
34	Stizostedion vitreum

It should be noted that continual updating will occur during the next several years as additional data points are entered into the initial data base. This means that the list will change due to additions (and probably a few deletions) and the predicted values will also change as the prediction equations become refined. Whereas no notice will be provided when the changes are made, the programs will be modified such that the operator will be automatically apprised as the changes will be reflected in the conversational aspects of the program. Figure 1 depicts how it works.

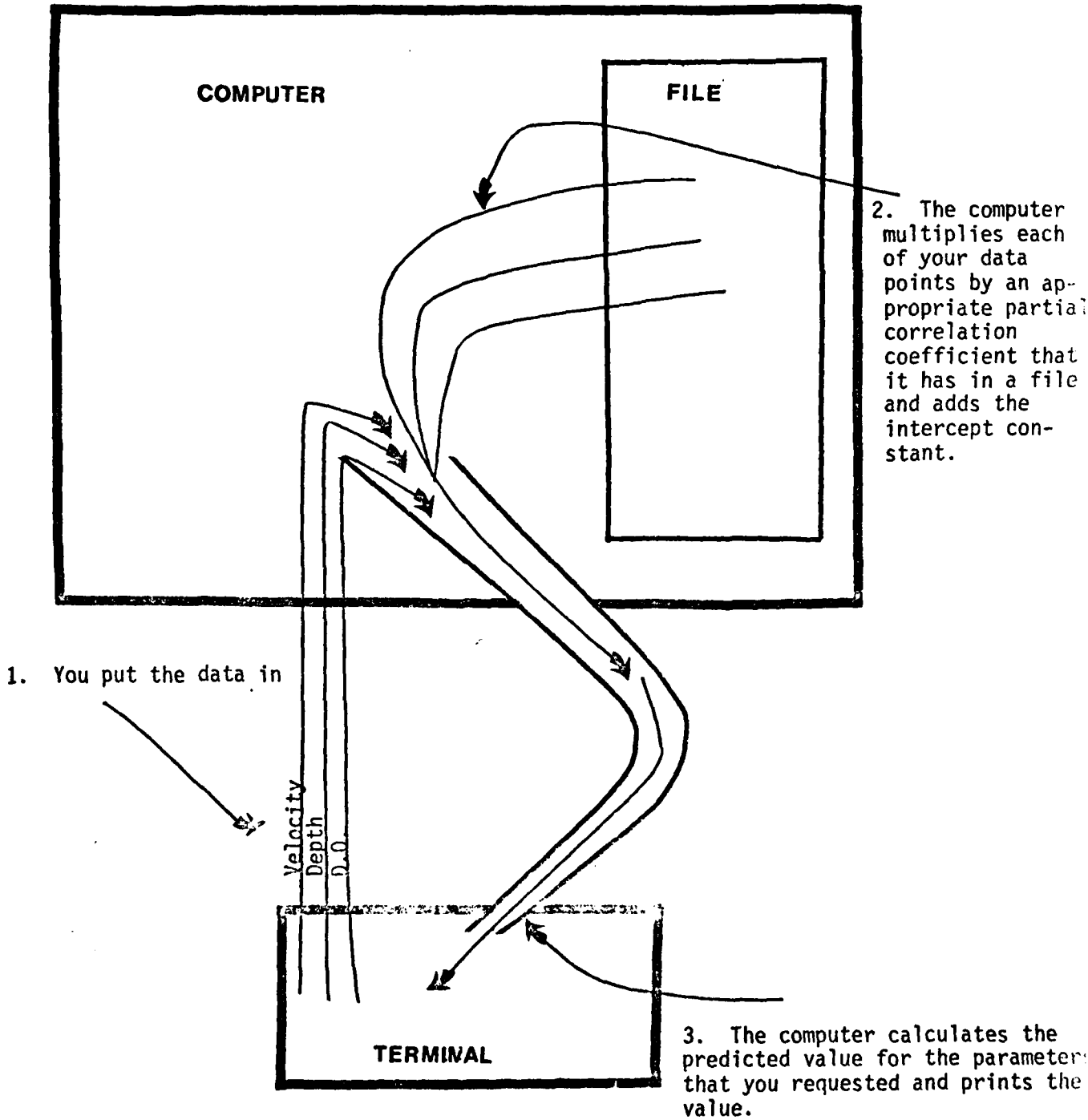
HOW CAN I USE THE MODEL? HOW CAN I ACCESS THE COMPUTER?

The model is currently stored in one of the computers on the UW-La Crosse campus, at La Crosse, Wisconsin. The program can be accessed from any terminal that is tied into the L.A.C.E. system (La Crosse Area Computers for Education) or it can be accessed by telephone from virtually anywhere in the country by long distance telephone. Once the program is accessed and instructed to run, the model is self-explanatory and self-instructive. (See Appendix I for a trial run example.)

WHAT TYPES OF INFORMATION MUST I GENERATE TO PERMIT THE MODEL TO MAKE A PREDICTION?

In order to arrive at a prediction, the computer will expect data to be supplied to it in a specific form. Obviously to predict an outcome, the operator must tell the computer what the affected area will probably look like after the proposed modification is made. <u>This means that estimates of post-change project data are required</u>. Since the change in the environment will not yet have been made, some careful field work will be

Figure 1. Conceptual drawing of information flow when using the simulation model.



required. This model uses the following parameters as predictor variables.

1. Current Velocity
2. Depth
3. Turbidity
4. Dissolving Oxygen
5. Temperature
6. Total Kjeldahl Nitrogen (Sediment)
7. Nitrate (Sediment)
8. Ortho-phosphate (Sediment)
9. Sediment Particle Size Distribution (6 Classes)

Because each parameter contributes to the predicted value, one should attempt to include estimates of all categories. However, the model will operate with missing data points.

Because of the complexities of the Mississippi River system, it is difficult to ascertain exactly what situations lend themselves to being modeled; however here are a few simple guidelines that might help: The model has been tested and appears to be accurate when:

1. The area to be affected is zoologically and botanically diverse.
2. The total benthic and macrophyte standing crops are high.
3. The perturbations are great; that is, the change in flow regimes and the associated changes in other parameters are significant.

Therefore, the model system does not have the ability to recognize small subtle changes in many cases. Since the predictor variable ranges are great, and the number of samples that represent the extremes are smaller, the accuracy of the predictions is better in the middle ranges. To demonstrate this from a practical point, then, here are a few instances

where the model should not be applied or at least relied upon to accurately predict:

1. In the main channel where both biomasses and diversity are extremely low and where the perturbation or change is slight.

2. In standing water areas where extreme eutrophy exists and the change or perturbation is slight (e.g. the diversion of 1-10 cfs).

3. In areas where the maximum values for the predictor variables in the original data base are exceeded (e.g. Lake Pepin).

Actually the model can be applied to any situation. However, the reliability of the predictions will vary according to the local circumstances encountered in the field. The best successes have been met in areas like Fountain City Bay where the inflow was significant and the receiving area was eutrophic. Since the environmental perturbation will cause some or all of these to change, the data contained in Table 2 will undoubtedly, be of some help. This table contains what are termed "most probable values" for all the above listed parameters. These were calculated by regression analyses. The values for each parameter were calculated with multiple regression analyses with current velocity as the independent variable. To see how these values might be employed, consider the shallow pond (previously discussed) which will receive a supply of flowing water due to the installation of a diversion structure. Several of the values for the required parameters can be identified. That is:

1. The depth of the receiving area can be measured. (One may have to speculate, however, concerning water elevation increases and scouring.)

2. The turbidity of the water supply can be measured.

3. The dissolved oxygen levels (measured at the sediment-water interface during the period of minimum oxygen levels) can be measured or extracted from the table.

4. The maximum summer temperature of the inflowing water can be measured.

Table 2. Most probable values calculated from regression analyses with current velocity as the independent variable, Pool 8.

CV	DEPTH	TURB	DO	TEMP	TKN	NO_3	NO_2	PO_4	S_1	S_2	S_3	S_4	S_5	S_6
0.000	---	8.40	0.40	--	4.135	.034	.011	.019	0.1	2.2	7.1	15.0	24.2	51.1
0.025	---	9.60	0.64	--	3.353	.032	.010	.018	0.2	3.1	7.3	15.0	24.0	50.0
0.050	---	10.80	0.88	--	2.567	.030	.010	.017	0.3	4.6	7.6	16.0	23.7	47.8
0.075	---	12.00	1.12	--	1.782	.028	.009	.015	0.4	5.4	7.7	17.0	23.4	45.5
0.100	---	13.20	1.72	--	0.997	.026	.008	.012	0.5	6.4	8.5	18.0	22.3	43.8
0.125	---	14.40	2.81	--	0.212	.024	.007	.010	0.6	7.3	9.3	20.7	21.1	41.0
0.150	---	15.03	3.71	--	0.197	.023	.006	.008	0.8	7.5	10.1	21.6	20.1	39.4
0.175	---	15.67	4.88	--	0.181	.022	.005	.007	0.9	7.7	11.4	23.0	19.4	36.4
0.200	---	16.30	5.91	--	0.173	.021	.005	.007	1.0	8.1	12.2	25.6	18.5	34.3
0.225	---	16.92	6.74	--	0.167	.020	.004	.006	1.2	8.7	13.5	27.4	17.0	32.2
0.250	---	17.55	7.01	--	0.161	.019	.003	.006	1.5	9.0	14.1	29.3	16.0	29.8
0.275	---	18.20	7.05	--	0.155	.018	.003	.005	1.8	9.4	16.4	31.0	15.0	27.0
0.300	---	18.81	7.09	--	0.149	.017	.002	.005	2.1	10.1	19.2	32.0	13.0	25.0
0.325	---	19.44	7.10	--	0.143	.016	.002	.005	2.2	10.8	20.6	32.0	12.4	22.0
0.350	---	20.10	7.11	--	0.137	.015	.001	.004	2.4	11.3	21.0	32.0	11.0	21.0
0.375	---	20.70	7.11	--	0.131	.014	.001	.004	2.6	11.9	22.1	33.0	10.0	19.0
0.400	---	21.30	7.11	--	0.125	.013	.001	.003	2.8	12.7	23.3	34.0	10.0	18.0
0.425	---	21.93	7.11	--	0.117	.012	.001	.003	3.0	13.8	24.2	34.0	9.0	17.0
0.450	---	22.56	7.12	--	0.111	.011	.001	.002	3.3	14.2	25.3	35.0	9.0	16.2
0.475	---	23.19	7.12	--	0.105	.010	.001	.002	3.6	14.9	25.4	35.0	8.0	15.0
0.500	---	23.82	7.12	--	0.099	.009	-0-	.001	4.0	15.6	26.0	35.0	8.0	13.0
0.525	---	24.45	7.12	--	0.092	.008	-0-	.001	4.4	16.1	26.1	35.0	7.1	12.0
0.550	---	25.08	7.12	--	0.084	.007	-0-	.001	4.8	16.7	26.5	35.0	7.1	10.0
0.575	---	25.71	7.12	--	0.077	.006	-0-	.001	5.2	17.2	27.0	35.0	6.0	9.0
0.600	---	26.34	7.12	--	0.068	.005	-0-	.000	5.7	17.6	27.5	35.0	5.0	7.0
0.625	---	26.97	7.12	--	0.061	.004	-0-	.000	6.3	18.0	28.2	35.0	4.0	6.0
0.650	---	27.60	7.12	--	0.055	.003	-0-	.000	7.4	18.6	29.1	35.0	3.0	4.0
0.675	---	28.23	7.12	--	0.047	.002	-0-	.000	7.9	19.2	29.7	35.0	2.0	3.0
0.700	---	28.86	7.12	--	0.039	.001	-0-	.000	8.2	19.9	30.4	35.0	2.0	2.0

5. The sediment nutrients can (and should) be measured prior to the diversion to determine what the initial levels are. Knowing that these will change with current velocity, the table can be consulted and those values can be used. A comparison of those to the pre-opening levels should be made however, to determine whether they are at least similar. This will at least determine whether the local situation is ordinately affected by large inflows on nutrients (such as sewage outfall etc.) This will help to validate the use of the nutrient values in the table.

6. The sediment particle size distribution can also be extracted from the table. Since the relationship between sediment fractions and current velocity have been established for this portion of the river. An examination of the sediment should also be made prior to the opening to assure that special circumstances do not exist (such as the presence of rip-rap or other artificial materials introduced to the area.)

The use of the table however, implies that you must project what the current velocity will be since the data in Table 2 are regressed against this variable. Current velocity, however, is one of the easiest to predict since you need only know:

1. The amount of water to be diverted, and

2. the cross-sectional area through which the water will flow.

Understandably, the flow of water into the recipient area will be neither uniform nor steady at all locations. Therefore, it would be useful to define a few options as to what might happen hydrologically in a given area and to calculate predictions for these options.

HOW SHOULD I SAMPLE THE AREA AND WHAT METHODS ARE REQUIRED TO MEASURE THE PREDICTOR VARIABLES?

Sampling an area for any biological population poses several problems, no matter how large or small or diverse the area is. Several methods of sediment sampling have been described in the literature. The field and laboratory methods that were utilized in this study are provided in Appendix II and are recommended for analyses. You may also consult the

literature cited in the bibliographical section of this manual for additional information.

I HAVE THE NECESSARY VALUES FOR THE PREDICTOR VARIABLES TO MAKE A RUN ON THE COMPUTER. NOW WHAT?

Now you need only access the computer, call up the appropriate program, and tell the computer to run. It will do the rest. That is, it will ask if you need a table of parameters to be predicted, and it will instruct you as to how you put your data into the program. When you have completed your routine, the computer will provide a predicted value for either the plant or animal that you choose from the table. Consult the example provided in Appendix II and follow those directions exactly. You may also consult Appendix III for additional information concerning programming and understanding computers utilizing BASIC language.

If you have missing data and fail to answer a request for a value, the program will assume that the value is zero. This will affect the prediction. Therefore the best predictions can be made from complete data.

APPENDIX I. REGRESSION MODEL - TRIAL RUN

Accessing the Computer

You may:

1. Call the River Studies Center, provide them with the input data, and request that they run the programs. The predicted values will be mailed to you after they have been calculated.

2. You may write to the River Studies Center, providing the center with appropriate data, and request that they run the programs to obtain the predictions.

3. You may dial the computer directly and execute the program yourself. Any teletype-compatible, ASCII coded, full duplex, 30 characters or less/second terminal with an acoustic coupler. This type of terminal can be purchased or leased from several computer equipment companies. Initial purchase costs range from approximately $700.00 to $1200.00.

If you choose option #3, you:

Dial (608) 785-8805 and listen for a high frequency signal.

With your terminal turned on and "on line", place the receiver in the acoustical coupler.

Press "line feed" and "return" in that order. The computer will type Please log on.

You type
HEL-R508,MISRIV (and return)

The computer will type:
R508 on port 00
READY

You type GET-BIOMOD.R504 (return) and
RUN (return)

The computer will type
BIOMOD, a regression model etc.

APPENDIX II. FIELD AND LABORATORY METHODS

Field Methods

Sediments

Sediment samples can be collected with either a core sampler or a suitable dredge such as a Ponar. Samples should be placed on ice until the laboratory analyses can be made.

Water

Water samples should be collected from the sub-surface and placed in acid-washed polyethylene bottles. Samples should be kept on ice until arrival at the laboratory.

Depth, Current Velocity, Discharge

Depth measurements can be made easily with a sounding pole in shallow water or a echo-sounding device in deeper portions of the pools. Cross sectional areas can be calculated by calculating the average depth of several soundings and multiplying the mean depth by the width of the channel. Discharge is the product of the cross-sectional area and the current velocity.

Current velocity is most accurately measured with a current meter. However, where flow is uniform, a floating object can be placed in the stream and the distance of travel can be timed.

Temperature and Dissolved Oxygen

Dissolved oxygen meters are available from several manufacturers. Most are equipped with a thermistor which facilitates easy calibration of the meter. Measurements of both temperature and DO are easily done with such an instrument.

Laboratory Methods

Sediments

In preparation for sediment particle size and sediment chemical analyses, the frozen samples should be thawed, dried at 106 C (APHA 1976) and then triturated with a porcelain mortar and rubber pestle.

Particle size analyses can be performed on a 50.0 g aliquot from each sediment sample. The material should be placed in a set of U.S.A. Standard Testing Sieves and then placed on a shaker for 30 minutes. The data should be reported as percent of each particle fraction. The screen sizes are noted in the computer program.

Total Kjeldahl nitrogen should be determined on duplicate aliquots (0.5 g) of each sediment sample using a suitable Kjeldahl technique (EPA 1974). Available sediment nutrients can be extracted from 2.0 g of sediment with continual agitation for 30 minutes with 225 ml of deionized water (Olsen and Dean 1965; Thomas and Peaslee 1973). The extract water is then vacuum filtered through prewashed glass fiber filters (Gelman Type A-E). The filtered extract is analyzed for dissolved ortho-phosphate (EPA 1974) nitrate (Barnes 1959) and nitrite (EPA 1974). Results should be expressed as mg N or P/g sediment.

APPENDIX III. UNDERSTANDING BASIC LANGUAGE AND THE TERMINAL

THE TERMINAL

Before you get under way, be sure you have familiarized yourself with your terminal. The terminal is the device through which you talk to the computer and by which the computer speaks with you. Your terminal may be either a printing device (rather like a typewriter) or a video display screen device (sort of a cross between a typewriter and a television).

Either type of terminal may be used with BASIC, provided you know how to do the following operations:

1. How to turn the terminal on and off.
2. How to establish a connection between the terminal and the computer.
3. How to enter data and complete an entry.
4. How to correct mistakes, for example -- erasing lines and backspacing over incorrect answers.
5. How to "break in" or override terminal activity in an emergency.
6. Where to find help when all else fails.

How to Turn the Terminal On and Off

Look for the switch on the keyboard, on the side casing, at the back, or even on the bottom of the terminal. Turn it to ON, or if there are options, turn the switch to the position called LINE. Be sure to turn the switch to OFF when you have finished using the terminal.

How to Establish a Connection

Once the terminal is on, press the keys marked RETURN and LINE FEED. If the terminal is connected directly, the computer will say PLEASE LOG ON. If the terminal is not connected, the computer will say nothing at all and you must phone the computer to establish the connection.

How to Complete an Entry

Every line of data you type must be terminated by the RETURN key. This key signals the computer that you have finished a line of input. The computer can respond by advancing the terminal to the next line permitting you to continue with your input, or even by asking you a new question. Remember -- a line of input may consist of only one word or character. You must still press the RETURN key to indicate that you are finished. A line or character may be erased only before the RETURN key is pressed. So be sure you like what you typed before you press the RETURN key and send it to the computer.

How to Correct Mistakes

If you realize that you've made a typing error and have not yet finished typing the line of data containing the error, you can erase the line by holding down the CONTROL key and the X key at the same time. The line is thus effectively "erased" and the terminal advances to a clean line so that you may try typing the data again.

```
THIS IS F\
THIS IS THE DATA
```

CONTROL/X prints a backward slash (\), erases the line, and advances the terminal.

A single character in a line can also be erased. Press the CONTROL key and the H key simultaneously to erase the last character you typed. Then type the character you really wanted in the first place. If you erred for more than just one character, hold down the CONTROL key and press the H key together for each character to be erased.

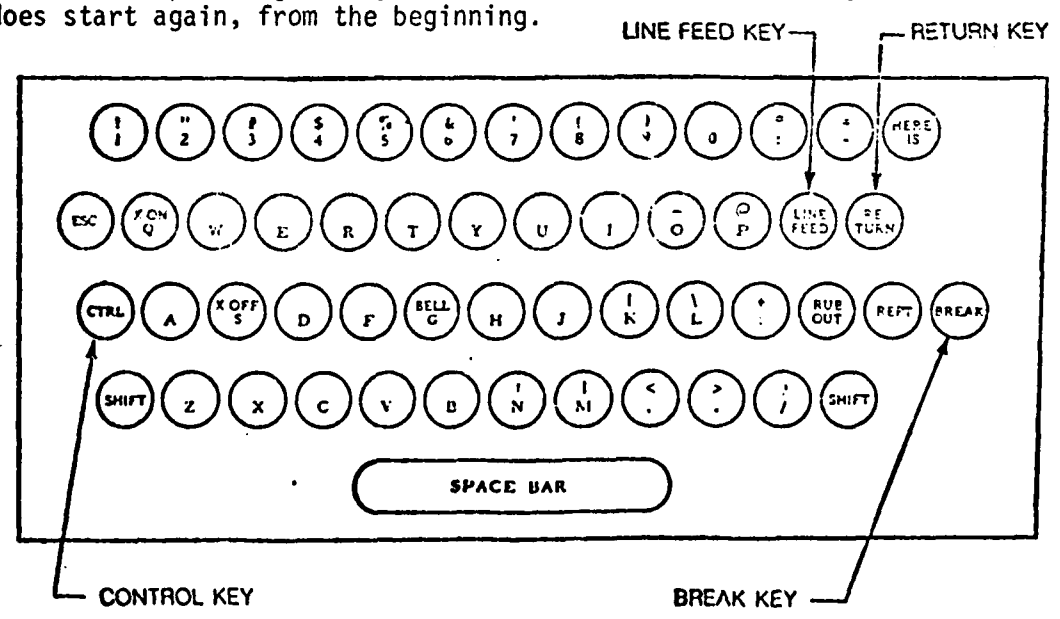
```
THIS IS THE DATT←A
```

Control/H prints an ← or __ and erases the last character.

Notice: The character keys used for correction may differ according to terminal. Experiment with your terminal until you are sure that you know how to erase lines and/or characters.

How to Break In

The BREAK key lets you stop a running (or runaway) program. This is an emergency button that lets you regain control of the terminal by interrupting a program when necessary. Of course, interrupting a program destroys what it was accomplishing, and you have to restart it by typing RUN. And it does start again, from the beginning.



LOGGING ON AND OFF

Logging on is the way in which you establish contact with the BASIC operating system. All you have to do to log on is to type HELLO and then identify yourself by user name and password.

```
HEL-R508,MISRIV
```

That's an example of logging on.

HELLO- is the command you must use. By the way it can be abbreviated to HEL- if you prefer.

R508 is a user name.

MISRIV is a password.

The dash (-) after HELLO and the comma (,) between the user name and password must be typed just as shown in the example. If you forget to include them, you won't be logged on.

Once you've introduced yourself, the computer begins to talk to you. This is what it will say.

```
R500 ON PORT #00, TIME, DAY, DATE, YEAR
L.A.C.E. U.W. LA CROSSSE 2000 SYSTEM I
READY
```

Having logged on, you may either create a program, or get an old one already stored in the computer and run it. To get a program you use the command GET- followed by the name of the program you want. To run the program type RUN.

```
GET-BIOMOD.R504
RUN
```

Whenever you'd like to terminate your working relationship with the operating system, just log off ... by typing

```
BYE
```

That's all it takes.

You can log off any time (provided a program isn't running). Just enter the command BYE, turn off your terminal and go home. But remember, once you've logged off you cannot speak to the computer again without first logging on. And, logging off erases any programs you might have been using or creating. So you have to start from scratch when you log on again.

A PROGRAM

A program is a set of statements. Each statement is an instruction to the computer. Each statement consists of one line of information and each is given its own number. Line numbers tell the computer the order in which the statements are to be performed. When you create a program, you enter statements (one at a time) at the terminal.

To enter a program, type:

```
10 PRINT 2+2
20 END

RUN
```

The computer then types

```
4

DONE
```

You entered and executed your program. But to clarify exactly what happened, let's do it one more time, a bit more slowly.

`10 PRINT 2+2` `20 END`	You typed your program, consisting of two lines. Line #10 said print the result of the arithmetic expression 2 plus 2. Line #20 said this is the end of the program.
`RUN`	Lastly, you executed or ran your program by typing the command RUN.
`4`	Performing the instructions in your program, the computer evaluated the arithmetic expression 2 plus 2 and printed the answer: 4.
`DONE`	The computer said it had finished with your program.

That is how the program was created and how it worked when it was executed.

Now, type RUN again.

```
RUN

4

DONE
```

The computer printed 4 again. That means that your program is still in memory within the computer and you can execute as often as you like.

Type RUN and see what happens.

If you should log off, you'll have to re-enter your program the next time you wish to use it since logging off removes a program from memory.

OPERATORS

BASIC uses symbols to define certain types of operations. BASIC also expects you to use these symbols (called operators) when defining operations. Each symbol corresponds to one operation such as an addition, subtraction, division, etc. Each symbol can be found on your terminal keyboard.

Sometimes a combination of keys are used to type a symbol. You may need to press the SHIFT key as well as the key containing the symbol, or you may need to enter two symbols together such as <=.

See if you can locate the following keys on your terminal. If there is no key containing a certain symbol, ask someone if there is another symbol you should use instead of the missing symbol.

Symbol	Definition	Example of Use	
=	equality	X+Y	A1=B2
+	add	2+2	X+Z
-	subtract	3-1	A-T
/	divide	4/2	Y/Z
*	multiply	9*3	M*N
#	inequality	Z#W	
<>	inequality	Z<>W	11<>23
>	greater than	4>3	Y>Z
<	less than	3<5	Z<Y
>=	greater than or equal to	Y>=X	X>=6
<=	less than or equal to	X<=Y	9<=Z
**	exponentiate	4**3	X**9
↑	exponentiate	4↑3	x↑9

INPUT

Data which is received by a program and processed in some manner is called input. Input can be received from terminal entry, from within the program itself or even from disc or tape files. As far as your programs are concerned, BASIC provides three different means of specifying input:

The LET statement.
The INPUT statement.
The READ statement.

Think of these statements as "doors" through which information can pass into your program for processing. Place these doors in your program wherever you want a data to be admitted. Look at the following program.

```
40 PRINT "X+Y=" X+Y
50 END
```

Enter it. You may use the program in the discussion of input and may add the input doors to it as you master them. You may also run the program from time to time, to see what changes the input doors have wrought.

The LET statement.

The LET statement is a handy way of inserting a value, or of grouping data under a common name. When used to insert data into a program, the LET statement tells the computer to let a variable equal the value you want.

To illustrate, if you want X to be equal to 27 in your program, type

```
20 LET X = 27
```

(20 LET 27 = X won't work. The symbol must come first.)

> Use the letters A,B ... X,Y,Z to represent items whose values can change. These are variables.

And if you want Y to equal 8, type

```
30 LET Y = 8
```

From this point on the letter X will be 27 and the letter Y will be 8 as far as your program is concerned.

List the program now and see what it says.

```
LIST

20  LET X=27
30  LET Y=8
40  PRINT "X+Y="X+Y
50  END
```

Now run it

```
RUN

X+Y= 35

DONE
```

The program added the value of the two variables and printed the result.

Using the LET statement to group data under a common name works as follows. You may simplify the PRINT statement by merging X+Y into a single character ... the letter Z. This is done with the statement

```
35 LET Z = X+Y
```

Next we change the PRINT statement to

```
40 PRINT "X+Y=" Z
```

Now to see exactly what we did to the program, let's list and run it.

```
LIST

20  LET X=27
30  LET Y=8
35  LET Z=X+Y
40  PRINT  "X+Y="Z
50  END

RUN

X+Y= 35

DONE
```

The Input Statement

The INPUT statement permits you to enter data for your program from your terminal while your program is executing. Here's how you use it. Insert an INPUT statement in your program at the point at which you want the program to stop and ask for data. When the program is run, its stops as soon as it encounters the INPUT statement and types a question mark (?) at your terminal. Data must then be typed at the terminal before the program can proceed. Each INPUT statement can ask for one or more variables.

Insert two INPUT statements in the program, one to ask for the value of X and one to ask for the value of Y.

```
20  INPUT X
30  INPUT Y
```

Now, list the program.

*Notice that the new lines numbered 20 and 30 replace the old LET statements that were created as lines 20 and 30 earlier.

Run it and see what happens.

```
RUN

?
```

The question mark asks for the value of the variable X and is printed by the computer in response to line 20 in the program. Give X the value of 106.

```
? 106
?
```

The second question mark is seeking the value of the variable Y (see line 30 in the program.) Type 94.

```
? 94
X+Y= 200

DONE
```

The computer added X and Y and printed the answer ... 200.

There is another way to specify variables with the INPUT statement. Multiple variables can be requested by a single INPUT statement. For example, you could have used the statement.

```
20 INPUT X,Y
```

When more than one variable is requested by a line, a comma must separate the variables.

In this case it is necessary to erase the now unnecessary line 30 by simply entering the line number and leaving the line blank.

```
30

LIST

20  INPUT X,Y
35  LET Z=X+Y
40  PRINT "X+Y="Z
50  END

RUN

?
```

Note. Line 30 is gone.

This question mark represents the values of both variables X and Y. Therefore, you must enter both.

```
??106,94
X+Y= 200

DONE
```

Once again our answer is printed by the computer.

Consider what would have happened if you had given only one answer, say 106, and then pressed RETURN. Try it.

```
RUN

?106
??
??94
X+Y= 200

DONE
```

This tells you that more input must be entered. If the INPUT statement expects two values, you must enter two values.

There are error messages or signals that appear sometimes when you are using the INPUT statement:

TRANSMISSSION ERROR, REENTER	This message means that the value was not correctly transmitted to the system. You must type it again.
BAD INPUT, RETYPE FROM ITEM xx	This message tells you that the value (xx) was not acceptable for the variable in questions. Type in the correct value, and any which followed it.

EXTRA INPUT, WARNING ONLY	This message is merely a warning. It means that extra values were included in the line. The computer ignores the extra characters, there is no need to re-type.

To get the last two error messages you had to violate some pretty tricky INPUT statement expectation. The INPUT statement determines the type of data to be entered, as well as merely requesting input.

Up to this point, you have been requesting number (numeric data). But you can request ASCII (character) data just as readily by adding a dollar sign to the variable. To illustrate

```
20 INPUT X$
```

Variables may be called any letter from A through Z or A$ through Z$. Variables from A0$ to Z0$ and A1$ to Z1$ may also be used.

Modify our program to request an alphabetic variable. Type and run

```
20 INPUT X$
35
40 PRINT X$

RUN

?X

X

DONE
```

The Read Statement

The READ statement has a partner -- the DATA statement. Together these two make it possible for you to place data in your program while you create it, and then to instruct the program to read the data at a specific point during execution.

Here's how these statements work. Insert these lines in our original input program.

```
10 READ X,Y
20 DATA 25,50
```

and list and run the program.

```
LIST

10  READ X,Y
20  DATA 25,50
40  PRINT "X+Y="X+Y
90  END

RUN

X+Y= 75

DONE
```

The data stored in a DATA statement is read from the left to the right by the READ statement. Thus in our program X was equal to 25 and Y to 50, and the total as pointed out by the computer was 75.

Now, let's add another kind of statement to our program.

```
50 GO TO 10
```

This statement tells the computer to go to line 10. Okay, list the program.

```
LIST

10  READ X,Y
20  DATA 25,50
40  PRINT "X+Y="X+Y
50  GOTO 10
90  END
```

Try to follow the flow of the program in your head. Pretend you are the computer:

```
 1a)  READ X - okay get the first number from the data statement   25
  b)  READ Y - get the next number from the data statement .....   50
 2a)  PRINT "X+Y=" ..................................... X+Y=
  b   X+Y - add 25 and 50 ..........................................  75
 3a)  GO TO 10 - go back to line 10 ..........................
 4a)  READ X - get the next number from the data statement .....
```

There isn't any next number in the data statement, we already used them all. In such a situation, the computer would print

```
OUT OF DATA IN LINE 10
```

and stop the program.

It is possible to add more data by adding more DATA statements, or to increase the amount of data in our original DATA statement. Try a little of both ... type

```
20
60 DATA 25,50,75,100
70 DATA 300,200
```

DATA statements can be placed anywhere within the program. The should probably be clustered together at the end of the program so that they can easily be found. (You may keep the data next to the READ statements if you wish, or group them all at the beginning. Since they can go anywhere, the choice is really up to you.)

Okay, let's list and run the program.

```
LIST

10  READ X,Y
40  PRINT "X+Y="X+Y
50  GOTO 10
60  DATA 25,50,75,100
70  DATA 300,200
90  END

RUN



X+Y= 75
X+Y= L75
X+Y= 500

OUT OF DATA  IN LINE  10
```

Check the answers by following the flow of the program as you did above.

There is another solution to the OUT OF DATA ... situation. You can use a RESTORE statement to tell the computer to re-use the same data over and over. Try it. Type

```
45 RESTORE
60 DATA 25,50
70
```

You have added a RESTORE statement to your program and, for convenience sake, gone back to your previously limited amount of data. Now, find the BREAK key on your terminal (you're going to need it!) Then list and run the program.

```
LIST
10  READ X,Y
40  PRINT "X+Y="X+Y
45  RESTORE
50  GOTO 10
60  DATA 25,50
90  END

RUN

X+Y= 75
X+Y= 75
X+Y= 75
X+Y= 75
X+Y= 75
X+Y= 75
```

The only way out is to hit the BREAK key ... so do it.

```
STOP
```

Granted, in your simple program, RESTORE did nothing more than create a loop from which you had to extract yourself rather awkwardly.

A SENSE OF DIRECTION

Remember your experience with the GO TO and RESTORE statements. You got into an endless loop from which the program could not escape. Well, it is possible to create useful loops in programs, and to insert controls within the program which terminate such loops at the appropriate moment. This is done with the IF ... THEN statement.

Take the program

```
10 LET X = 5
20 PRINT X
30 LET X = X+5
40 GO TO 20
50 END
```

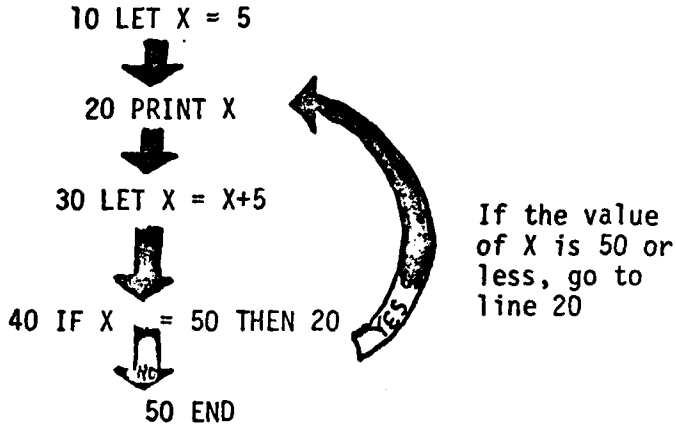
If you ran the program as it stands, you would once again find yourselves in an endless loop ... in this case, counting by 5's ad infinitum.

But if you substitute an IF ... THEN statement for the GO TO statement, you set an upper limit on that counting process and, in effect, tell the program when to stop counting.

Try counting to 50 by adding

```
40 IF X< = 50  THEN 20
```

Here's how the statement works. It tells the computer to make a decision based on the value of X, and where to go as a result of that decision. To illustrate:



Run your program and see if it functions as anticipated.

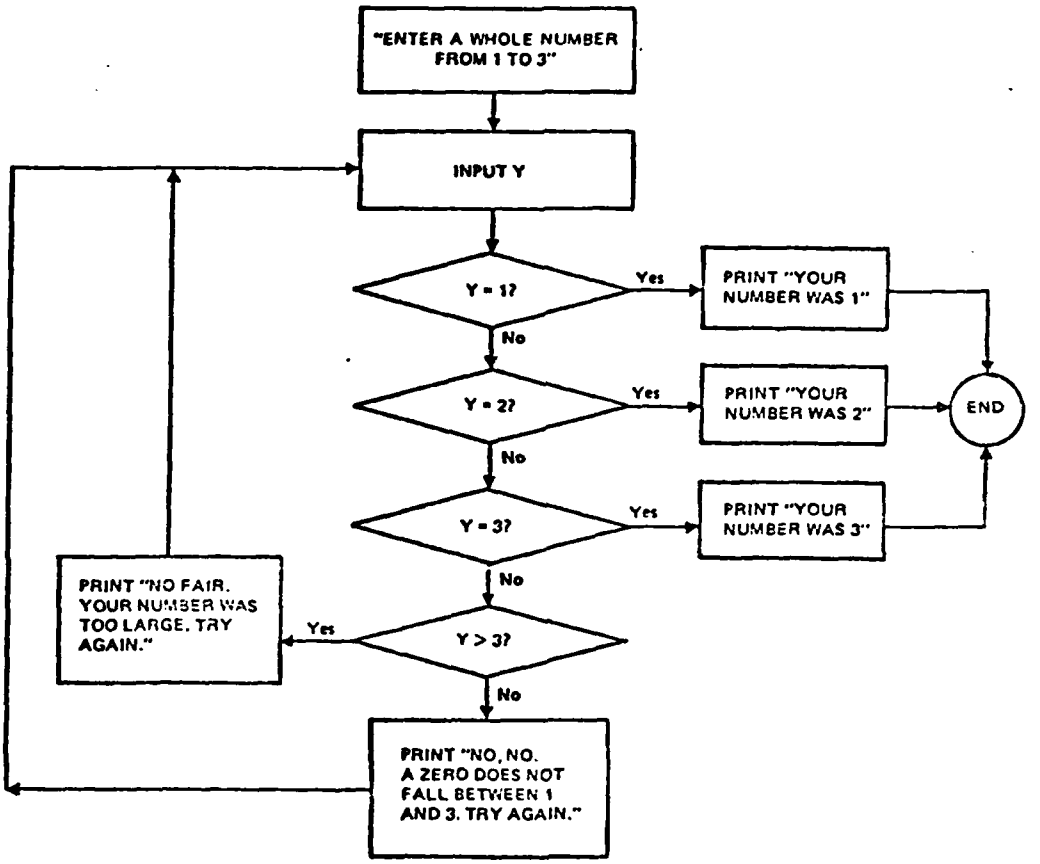
```
RUN

5
10
15
20
25
30
35
40
45
50

DONE
```

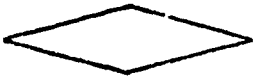
You can use other operators in your program if you wish. Try using the symbols for less than or greater than in the IF ... NEXT statement and see if you can predict the results.

"What about using GO TO? Didn't we just obsolete it with the IF ... THEN business?" No. The GO TO statement is still a valid and, indeed, useful statement. Here is how both the GO TO and IF ... THEN can be used together to create a program.

```
5  PRINT "ENTER A WHOLE NUMBER FORM 1 TO 3"
10 INPUT Y
15 IF Y = 1 THEN 45
20 IF Y = 2 THEN 55
25 IF Y = 3 THEN 65
30 IF Y > 3 THEN 75
35 PRINT "NO. A ZERO DOES NOT FALL BETWEEN 1 AND 3.  TRY AGAIN."
40 GO TO 10
45 PRINT "YOUR NUMBER WAS 1"
50 GO TO 90
55 PRINT "YOUR NUMBER WAS 2"
60 GO TO 90
65 PRINT "YOUR NUMBER WAS 3"
70 GO TO 90
75 PRINT "NO FAIR.  YOUR NUMBER WAS TOO LARGE.  TRY AGAIN."
80 GO TO 10
90 END
```

If you enter and run this program, you will discover that the one way to make the program come to a normal end is to do as it says and enter a 1, 2, or 3 when it tells you to. Entering any other whole number will send you back to the beginning of the program. The program's processing flow is depicted below.





is a decision box. It indicates that the computer must decide *yes* or *no* to a situation, and proceed accordingly.

is an operation box. It indicates that the computer must perform the task stated in the box and then proceed to the very next box (operation).

P.S. Flow charts are invaluable programming aids. They help you organize your thoughts before writing a program -- and they help you figure out what a program's doing (if there's any doubt) after you've created it. Get into the habit of using flow charts. You'll be glad you did.

The computed GO TO statement. A computed GO TO frees you from repetitive GO TO statements by letting the computer do some of the work of directing a program.

On your program you are going to replace lines 15, 20, and 25 with one line -- a computed GO TO statement. So instead of this

```
15 IF Y = 1 THEN 45
20 IF Y = 2 THEN 55
25 IF Y = 3 THEN 65
```

You have this

```
20 GO TO Y OF 45,55,65
```

which means the very same thing:

If Y is 1, go to the first number, in this case 45.
If Y is 2, go to the second number, 55.
If Y is 3, go to the third number, 65.
If Y is none of these, continue with the next statement.

Don't forget to erase lines 15 and 25 when you insert the computer GO TO statement.

OUTPUT

Output is what your program gives back to you at the end of or even in the midst of execution. Output is literally anything printed by the program. The output statement with which we shall be concerned is the PRINT statement. The PRINT statement tells the computer to type a value at the terminal. A value may be a number or a word, a letter or a sentence, or the result of an expression. The PRINT statement also monitors the spacing of the values that it prints across the page at the terminal.

If a single value is involves, it is printed and the terminal automatically moves to the next line. Remember your first program?

```
10 PRINT 2+2
20 END
```

That's what it did. It simply printed the result of the expression 2+2 and advanced to the next line.

Well, it is also possible to use multiple values in a PRINT statement, provided they are separated by commas or semicolons.

```
10 PRINT 5,10,15,20,25
20 END

RUN

 5          10          15          20          25

DONE
```

NOTICE that when a comma is used to separate the values in a PRINT statement, they are widely spaced across the page, beginning in columns 0,15, 30,45 and 50 unfailingly. Also note, 5 values on one line are the limit allowed with commas as separators.

Now then, try semicolons.

```
10 PRINT 5;10;15;20;25;30;35;40
20 END

RUN

 5  10  15  20  25  30  35  40

DONE
```

Two things changed. First, move values could be used (up to 12) and second the values were printed more concisely across the page.

When a value is enclosed in quotation marks, it is printed as is. Quotation marks are used primarily for alphabetic characters and special characters ... not numbers. To illustrate

```
10 PRINT "COLUMN A", "COLUMN B", "COLUMN C"
20 PRINT 120, 230, 820
30 PRINT 4.6901,333.9
50 END

RUN

COLUMN A        COLUMN B        COLUMN C
 120             230             820
 4.69            1               333.9

DONE
```

To control the placement of values across a line, use the control functions TAB, SPA, or LIN. TAB causes the carriabe to tab over to the column specified. Change line 20 in your program to read

```
20 PRINT TAB (2);120;TAB(19);360;TAB(34);820
```

Note the placement of semicolons. They must follow each TAB and value.

SPA causes the carriage to skip the number of columns specified. Change line 30 to

```
30 PRINT SPA (3),4.690;TAB(21);1;TAB(34);333.9
```

SPA uses a comma or a semicolon.

LIN generates a carriage return and, if specified, linefeeds. LIN(0) produces a single carriage return and zero linefeeds. While LIN(10) produces a single carriage return and 10 linefeeds. Add

```
35 PRINT LIN(2)
40 PRINT "THAT'S ALL"
```

List and run the program.

```
LIST

10  PRINT "COLUMN A", "COLUMN B", "COLUMN C"
15  PRINT
20  PRINT TAB(2);120;TAB(190;360;TAB (34);820
30  PRINT SPA(3),4.69;TAB(21);1;TAB(34);333.9
35  PRINT LIN (2)
40  PRINT "THAT'S ALL"
50  END

RUN

COLUMN A        COLUMN B        COLUMN C
  120                360             820
    4.69               1          3333.9

   THAT'S ALL

DONE
```

FILES

From time to time you may find it convenient to save the results of one program for use as input to another program. This is done by means of BASIC files.

To use BASIC files your must know how to use the following commands and statements:

- CREATE-
- PRINT
- READ

The first thing you must do is to create a file. While you create a file, you give it a name and a length, and place it in your library for storage. Let's create a file two records long called FILEA. Type

```
CREATE-FILEA,2
```

The number 2 means that your file contains two records. In serial data files each record will hold 123 data points.

You now have a file in our library called FILEA and capable of holding two records. It's empty at this point. Now, you are going to write two programs ... one to write data into the file, and a second to read data from it. For the program, type

```
10 FILES FILEA
20 LET X =9
30 LET Y =1
40 PRINT #1; X+Y
50 PRINT #1; Y-X
60 END
```

The files statement identifies the file to receive the data.

The #1 added to the print statements tells the computer to print to the first file in the FILES statement. You defined only one file in the FILES statement, but we could have listed as many as 16.

Run the program

```
RUN

DONE
```

That, essentially, is how to create and use files with your program.

Once you have finished with a file, you should remove it from your library. You do this by purging the file. Simply type the command PURGE, a hyphen,

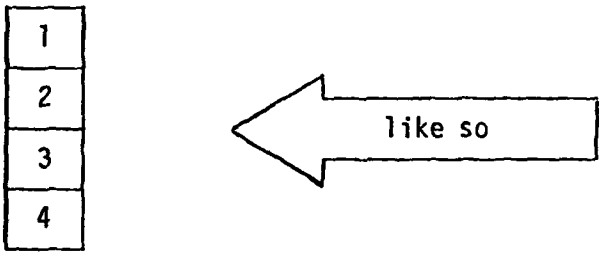
and the name of the file you want removed. To remove our file above, we would type

```
PURGE-FILEA
```

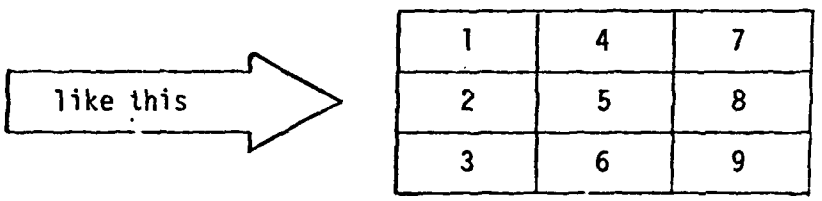
AN ARRAY IS

A group of data identified by a common name. The data in an array is divided into units called elements.

An array may consist of one dimension which provides a list of elements



Or an array may be two dimensional, providing a table of elements



Each array must be given a one character name (A through Z). A single character is also used to identify a variable. Just like string variables a DIM (dimension) statement must be used to define an array.

Type

```
10 DIM A(4,3)
```

You just created an array named A having the dimension 4 by 3. That's four rows by three columns.

Now, let's get some data into the array. Type

```
20 MAT READ A
30 DATA 1,2,3,4.5,6,7,8.9,10,11,12
40 MAT PRINT A
90 END
```

The MAT READ statement tells the computer to take the data from the DATA statement and place it in the array. The data is inserted into the array in a predicatable sequence ... row by row.

The PRINT statement will print out the array. Run the program and see how the array looks.

```
RUN

 1            2            3

 4            5            6

 7            8            9

10           11           12
```

Compare the DATA statement and the sequence of data in the array. That is the order in which data will always be stored in an array, and that's important to know.

Let's say you want to print only the element containing the number 8. How would you tell the computer to do this? With a PRINT statement and with the element's dimensions (row and column). The 8 is in row 3 column 2, so we would say

```
40 PRINT A(3,2)
```

Replace the first PRINT statement with the one above and run the program.

```
RUN

 8

DONE
```

Note: Use MAR PRINT to print the whole array, and use PRINT to print an element.

So what you learned about arrays? You have learned some new statements -- MAT READ and MAT PRINT. Or so it seemed. But think about it. These are actually the old familiar READ and PRINT statements with MAT added on in front. The MAT tells the computer that you are referring to an array. MAT stands for matrix which is another word for tabular array. (You might also find arrays called subscripted variables in some books.)

Writing a program by designing the output is not really working backward as you might initially have thought. Ninety-nine percent of all programming is accomplished just that way. You start with the output you want, evaluate the input you have, and devise a means of converting your input into output. The means you devise is, of course, the program.

There are several programming aids that are worth introducing here: coding sheets and formatting forms. Coding sheets enable you to write down all the things you need to tell the computer before you are actually sitting at the terminal. Formatting forms come into play even earlier. They are used to design the output (report usually) before you even begin coding.

To review then ... the recommended sequence of events for creating a program is:

1. Design your output. Lay it all out on a formatting sheet, deciding on columns, headings and the like at this point.

2. Flow chart. Outline your plan of action, going from each bit of available input to each point of final output. Make your decisions here. Find the shortest and surest path for each item.

3. Code up your plan using the appropriate BASIC statements.

4. Enter your program at the terminal. Then list and run it, and if necessary correct your coding or entry errors.

By now you should have the program you set out to create. And it should execute just as you anticipated.

APPENDIX IV: PERTINENT LITERATURE

PERTINENT LITERATURE

American Public Health Association. 1976. Standard methods for the examination of water and wastewater. 14th ed. APHA, New York. 1193 pp.

Anderson, H. W. 1976. Reservoir sedimentation associated with catchment attributes.

Barnes, H. H. 1959. Inorganic nitrogen: nitrate: Pages 113-125 in Apparatus and methods of oceanography. Interscience Publishers, Inc., New York.

Bhutani, I. J., R. Holberger, W. E. Jacobson, P. Spewak, and J. B. Truett. 1975. Impacts of hydrologic modification on water quality. Nat. Tech. Inform. Service, Springfield, VA (Pub. PB-248 523).

Burks, B. D. 1953. The mayflies, or ephemeroptera, of Illinois. Illinois Natural History Survey Bulletin. 26:1-216.

Claflin, T. C., R. G. Rada, and D. J. Grimes. A predictive regression model for Navigation Pool No. 8, Upper Mississippi River. Report submitted to GREAT I - USFWS, Twin Cities, MN. 583 pp.

Cochrane, T. 1976. Preliminary reports on the flora of Wisconsin as of January, 1976. 8 pp mimeographic reproduction, Herbarium, Department of Botany, University of Wisconsin, Madison, WI.

Cooley, W. W. and P. R. Lohnes. 1971. Multivariate data analysis. Wiley and Sons, New York. 364 pp.

Crabbe, J. A., A. C. Jermy, and J. T. Mickel. 1975. A new generic sequence for the pteridophyte herbarium. Fern Gazette. 11:141-162.

Cronquist, A. 1968. The evolution and classification of flowering plants. Houghton Mifflin Co., New York.

Eddy, S. and A. C. Hodson. 1957. Taxonomic keys to the common animals of the north central states exclusive of the parasitic worms, insects, and birds. Burgess Publishing Company, Minneapolis, MN.

Edmondson, W. T. 1959. Freshwater Biology. 2nd ed. J. Wiley & Sons, Inc., New York. 1248 pp.

Environmental Protection Agency. 1974. Methods for chemical analysis of water and wastes. USEPA, Washington, DC. 298 pp.

Fassett, N. C. 1957. A manual of aquatic plants (with revision appendix by E. C. Ogden). University Wisconsin Press, Madison, WI.

Gleason, H. A. 1952. New Britton and Brown illustrated flora of the northeastern United States and adjacent Canada. 3 vols. Lancaster Press, Lancaster, PA.

Gleason, H. A. and A. Cronquist. 1963. Manual of vascular plants of northeastern United States and adjacent Canada. Van Nostrand Co., New York.

Heineman, H. G. and D. L. Rausch. 1976. Distribution of reservoir sediment - Iowa and Missouri deep loess hills. Proceedings of Third Fed. Inter-agency Sed. Conf. March 22, Denver, CO. Water Resources Council, Washington, DC.

Hilsenhoff, W. L. 1975. Aquatic insects of Wisconsin: generic keys and notes on biology, ecology, and distribution. Wisc. Dept. Nat. Res. Tech. Bull. No. 89. 52 pp.

Hurst, A. J. and P. C. Chao. 1974. Sediment deposition model for the Tarvela Reservoir. Symposium on modeling techniques, Vol. I, 2nd Ann. Symp. on Waterways, Harbors and Coastal Engin., Sept. 3-5, San Francisco, (ASCE).

Liddicoat, M. K., S. Tibbitts, and E. I. Butler. 1975. The determination of ammonia in seawater. Limnol. & Oceanogr. 20(1):131-132.

Mason, W. T., Jr. 1973. An introduction to the identification of chironomid larvae. Analytical Quality Control Laboratory, EPA, Cincinnati, OH. 90 pp.

Merritt, R. W. and K. W. Cummins, eds. 1978. An introduction to the aquatic insects of North America. Kendall/Hunt Publishing Co., Dubuque, IA. 441 pp.

McHenry, J. R. 1974. Reservoir sedimentation. Wat. Res. Bull. Vol. 10:2, pp. 329-337.

McKee, G. D., L. P. Parrish, C. R. Hirth, K. M. Mackenthun, and L. E. Keup. 1970. Sediment-water nutrient relationships. Water and Sewage Works. June. pp. 203-206.

Mortimer, C. H. 1941. The exchange of dissolved substances between mud and water in lakes. Ecology. 29:280-329.

Olsen, S. R. and L. A. Dean. 1965. Phosphorus. Pages 1035-1048 in C. A. Black, ed. Methods of soil analysis. II. Chemical and microbiological properties. American Society of Agronomy, Inc., Madison, WI.

Pennak, R. W. 1953. Fresh-water invertebrates of the United States. Ronald Press, New York, NY. 769 pp.

Shaw, S. P. and C. G. Fredine. 1971. Wetlands of the United States: their extent and value to waterfowl and other wildlife. USFWS Circular 39. 68 pp.

Smart, Miles M. 1977. Nitrogen and phosphorus cycling by Nymphaea tuberosa and Ceratophyllum demersum in Lake Onalaska, Navigation Pool 7 of the Upper Mississippi River. MS Thesis, Univ. Wisconsin-La Crosse, WI.

Strodthoff, K. M. 1978. Nitrogen and phosphorus exchanges between water, sediment, and macrophytes in Lake Onalaska of Pool No. 7, Upper Mississippi River, 1975 to 1976. MS Thesis, Univ. Wisconsin-La Crosse, WI.

Swanson, S. D. 1978. The flora of the Upper Mississippi River floodplain project status report IV. Contr. Herbarium Univ. Wisconsin-La Crosse. 23:32 pp.

Swanson, S. D. 1977. The flora of the Upper Mississippi River floodplain project status report III. Contr. Herbarium. Univ. Wisconsin-La Crosse. 19:17 pp.

Swanson, S. D. and S. H. Sohmer. 1978. The vascular flora of Navigation Pool 8 of the Upper Mississippi River. Proc. Iowa Acad. Sci. 85: 45-61.

Thomas, G. W. and D. E. Peaslee. 1973. Testing soils for phosphorus. Pages 97-114 in L. M. Walsh and J. D. Beaton, eds. Soil testing and plant analysis. Soil Science Society of America, Inc. Madison, WI.

U.S. Dept. of the Army, St. Paul District Corps of Engineers. 1969. Master regulation mannual for (the) Mississippi River nine foot channel navigation projects. 78 pp.

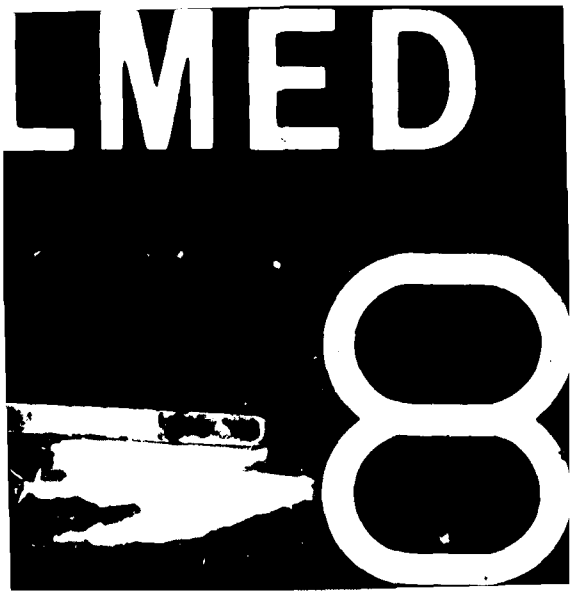
USGS. 1972. Cochrane Quadrangel, Wisconsin-Minnesota, 7.5 minute series (topographic) map.

Usinger, R. L., ed. 1956. Aquatic insects of California. University of California Press, Berkeley, CA. 508 pp.

Voss, E. G. 1972. Michigan Flora: Part I: Gymnosperms and monocots. Cranbrook Institute Science, Bloomfield Hills, MI. Bulletin 55: 488.

Westlake, D. F. 1963. Comparisons of plant productivity. Biol. Rev. 38:385-425.

Wiggins, G. B. 1977. Larvae of the North American caddisfly genera (trichoptera). University of Toronto Press, Toronto. 401 pp.


LMED
8